ATARI ACSI/DMA Integration Guide

June 28, 1991

A Reference Manual on the **ATARI** Computer Systems Interface (ACSI). Includes block diagrams, timing diagrams, command syntax, cable pinouts and interface information on peripheral devices currently adapted to the ACSI.

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1: INTRODUCING DMA

DMA Definition and General Description

When used in this manual "**Atari ST**" includes the ST, Mega, STE, Mega STE, TT030 and any future Atari computers that support the Atari ACSI bus. The term "**host**" may be used interchangeably with "Atari ST".

Refer to the "TT030 TOS Release Notes" and the "Atari TT030 Hardware Reference Manual" for specific information regarding DMA on the TT.

DMA is an acronym for Direct Memory Access. When used in the context of a computer (such as the Atari ST) and its high performance peripherals (floppy and hard disks, laser printers, tape drives, CD ROMs, etc.), DMA refers to the direct transfer of data between the peripheral device and the computer's memory. This transfer is done without the direct intervention of the processor. The Atari ST provides an external DMA bus connection through the Atari Computer System Interface (ACSI) connector on the Atari ST computer.

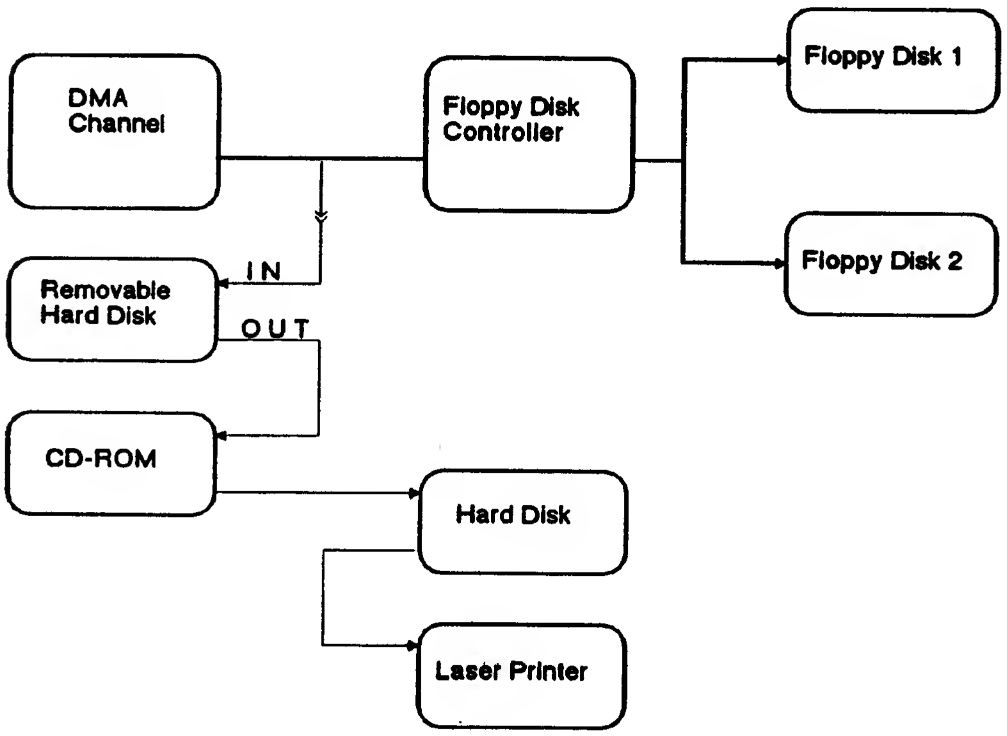
When the program running in your Atari ST computer requires data from a peripheral connected to the DMA channel (directly or through the ACSI bus) or needs to send data to the device, the program must perform the following basic steps. Refer to the chapter 4 section called **DMA PROGRAMMING STEPS** for detailed instructions.

1. Set a starting Atari ST memory location for the DMA data to be sent to or received from.
2. Set a DMA count. This count is (at least) the integral number of 512 byte blocks to be sent to or received from Atari ST memory.
3. Set the direction of data flow. This is accomplished by setting the DMA Read/Write bit to 0 for a read from a peripheral device and 1 for a write to a device.
4. Write the command to be performed to the peripheral device and wait for command completion.
5. Select the DMA source (external or internal). DMA takes place here, with the peripheral indicating completion.
6. Check the device status for an error. If no error occurred the data is now correctly placed in the peripheral or Atari ST memory (depending on whether you were sending it to, or receiving it from, the peripheral).

Peripherals Normally Connected to the DMA Port

As of the publication date of this manual, Atari ST peripheral devices that use the DMA channel include the Atari ST floppy disk controller (which is connected directly to the DMA channel) and the following devices which attach to the ACSI bus: the Laser printer through its APPC (Atari Page Printer Controller), the Hard Disk through its AHDI (Atari Hard Disk Interface), the CDAR Audio/ROM CD Unit, and the Removable Hard Disk. Other devices may be made available by Atari or third parties.

General Atari ST DMA/ACSI Block Diagram



CHAPTER 2: INTRODUCING THE ACSI PERIPHERAL INTERFACE

ACSI Definition and General Description

ACSI is an acronym for **Atari** Computer Systems **Interface**. This peripheral bus is very similiar to the industry standard **SCSI** (Small Computer Systems Interface) bus. A peripheral bus allows more than one device to be connected to the computer by way of standard signal lines.

The ACSI is an 8 bit bidirectional data bus that allows data to pass from a peripheral, such as your hard disk, to the host Atari ST computer. The transfer of data may also flow over the bus from the Atari ST to the peripheral. Data may be transfered at a maximum rate of 1.25 million bytes per second and may be transfered to or from any of eight possible control devices attached to the bus. These control devices are addressed as devices 0 to 7.

NOTE: Computer Peripheral: hardwired or DB19 socket (female)
Cable: DB19 Plug

ACSI Bus Signals and Descriptions

Data (D0-D7)	These 8 active high signals are the data bits.
Data Request (_DRQ)	Used by the peripheral to request a DMA data transfer. This line is used to request a byte of data during the DMA data transfer. The direction of the transfer is identified by the R/_W signal which is controlled by the DMA Read/Write bit in the DMA Mode Control Register. _DRQ is driven by the peripheral with an open collector output. Only the peripherals that have been commanded by the host peripheral may drive _IRQ, _DRQ, or the data bus. A peripheral may not spontaneously gain the host's attention.
Chip Select (_CS)	The Atari ST is writing a command or data byte to the peripheral, or is reading a handshake or status byte from the peripheral. When A1 and R/_W are low, asserting _CS (low) alerts all ACSI peripheral devices that the first byte of a command packet is presented to the bus. The first byte contains the device's operation code and the ACSI device number. __CS (low), R/_W (low) with A1 high, is then asserted for each of the remaining command bytes. Upon completion of a command (generally signalled by receipt of another interrupt from the device) the device status (generally 1 byte) is then read by the Atari ST by asserting _CS while A1 is high. Most standard commands are six bytes in length, but only the Inquiry, Read, and Test Unit Ready commands are required to be six bytes, in order to maintain consistency for the identification of and "booting" from different types of device.
Peripheral Acknowledge (_IRQ)	This active-low signal is driven by the target peripheral to indicate either (a) its readiness to accept another command byte, or (b) the availability of a byte to be read. All peripherals should de-assert _IRQ as soon as possible, with a maximum time of 20 microseconds of the next falling edge of _CS (and ideally should do it in under 500 ns).
Address 1 (A1)	When low during a _CS write, this signal indicates that the first byte of a new command (which contains the device selection ID) is being presented by the Atari ST. All other _CS style reads and writes should have A1 high.
Read/_Write (R/_W)	Used to determine direction of data transfer over the ACSI bus. Low signifies flow from the Atari ST to the peripheral device, i.e. Write. High signifies flow from the peripheral device to the Atari ST, i.e Read.

Host Acknowledge (_ACK)	The Atari ST uses this signal to notify the peripheral that the Atari ST has received a DMA'd data byte, or that data is being presented for the peripheral. This is a host-driven response to the _DRQ signal generated by the peripheral device. The signal is used by the Atari ST to (a) signify to a peripheral device that a byte of DMA data has been received by it, or (b) to respond to the device's _DRQ, indicating that the host computer is presenting DMA data.
Reset (_RST)	Reset all peripherals. Resets the ACSI bus and clears all occuring or pending operations. This signal has the highest bus priority (takes precedence over all other bus signals) and is the system-wide RESET signal. The Reset signal should only be used as an input to peripherals.

NOTE: Only peripherals that have been selected by the host sending them a command may drive _IRQ, _DRQ, or the data bus. A peripheral may not spontaneously gain the host's attention.

ACSI and SCSI - The Differences

Some important differences between ACSI and SCSI:

Physical cable connection - ACSI uses a 19 pin shielded cable versus the 50 pin flat ribbon cable used by SCSI.

The master/slave relationship - This defines who is in charge of bus operations between the computer and its peripherals. The ACSI bus specifies that the host computer is always in control of the bus as opposed to SCSI which allows both computers and peripherals to initiate control of the bus .

Maximum cable length - ACSI allows a maximum of 2 feet between devices with a total length allowable of 88". However, total cable length should be minimized in the interests of signal reliability. SCSI has a total allowable length of approximately 18 feet.

Maximum Number of Devices - System reliability requires that no more than four ACSI peripheral controllers be daisy-chained to an Atari ST.

Utilization of the SCSI Message phase (extended status) - SCSI allows it and ACSI doesn't.

Connect and Disconnect - the ability for a device to disconnect after receipt of a command and connect only when data is to be transferred. ACSI does not utilize this, SCSI can.

Bus Termination-SCSI uses open-collector bus drivers with 220/330 ohm termination resistors. ACSI uses CMOS or LSTTL level interfacing without terminating resistors.

Request/Acknowledge-There is an important conceptual difference between ACSI and SCSI regarding the Request/Acknowledge handshaking. SCSI uses a simple REQ/ACK procedure, whereas ACSI has CS/IRQ and DRQ/ACK procedures. See "Command/Status/Data Transfers" below for a detailed explanation of this difference.

Maximum Command Number-Because the ACSI controller device number is encoded in the most significant three bits of the command byte, the device can only use commands in the range $00-$1F.

Command/Status/Data Transfers-SCSI transfers all Commands, Status or Data information using its REQ/ACK (Request / Acknowledge) handshake. Whether this is done in a "DMA" mode or with byte by byte transfers through the controller processor, is a function of the SCSI controller used. ACSI provides for two methods of transferring information between the host processor and peripheral:

1. **Software Handshake Method** where each byte is transferred under processor control. This method uses the _CS and _IRQ signals. The processor asserts _CS to transfer a byte and the peripheral asserts _IRQ to notify the processor that it is ready to receive or send the byte. The direction of the transfer is determined by the R/_W signal. This handshake sequence is always used for sending commands and receiving the single byte of status issued upon error or command completion. This method <u>should</u> <u>not</u> be used for data transfer.

2. **DMA Method** where the processor issues a command (using the **Software Handshake Method** above) specifying the peripheral device involved in the data transfer, the type of command, the maximum number of blocks (512 bytes) to be transferred. Note that DMA time-shares the bus with the processor - only taking the bus when it needs to fill/empty its FIFOs. The FIFOs minimize the overhead involved in obtaining and releasing the bus. Transfer is considered complete when all bytes are transferred (i.e. the DMA count register == 0) or when an error occurs. This method uses the _DRQ/_ACK handshake for each byte transferred under DMA control. This is a very fast hardware handshake that occurs without any intervention from the processor. _ACK is a response to the peripheral device generated _DRQ signal and signifies that the host has received a byte or has one ready to send. _DRQ is asserted by the peripheral device when it is ready to send or receive a byte.

CHAPTER 3: UNDERSTANDING THE HARDWARE

Hardware Overview - DMA

A direct main memory access channel is provided to support 8 bit device controllers. The base address for the DMA read or write operation is loaded into the DMA Base Address Registers. Only one channel is provided, therefore only one DMA operation can be executed at a time.

The actual DMA operation is performed through two16 byte FIFO buffers. The DMA controller is programmed via the DMA Mode Control Register (16 bit word wide, write only, not reset by system reset) and DMA Sector Count Register (16 bit word wide, only the least significant byte of which is currently implemented). The success or failure of a DMA operation may be reported through the DMA Status Register (16 bit word wide, read only.) (However, this register is rarely used; usually a peripheral interrupts to indicate completion.) Changing the state of DMAOUT in the DMA Mode Control Register will reset the DMA controller.

Hardware Overview - ACSI

The ACSI commands are very similar to the SCSI commands. Each command is sent as a command packet and is typically 6 to 10 bytes long. The command packet length is determined by the requirements of the peripheral controller. For programming example purposes in this document, ACSI commands are assumed to have a length of six bytes.

NOTE: In actual use commands other than the Inquiry, Read, or Test Unit Ready command (for bootable devices) may have more or less than six bytes. The Inquiry command must always have six bytes for consistency. The Read command, when used with bootable devices, must have six bytes to be compatible with the Atari boot routines.

The first byte contains the command opcode in the lower five bits and the ACSI unit number in the upper three bits. Each of the eight potential devices on the ACSI bus has its own unique device number (0-7) which is usually set by a dip-switch on the controller board. A device should only respond to commands sent with its controller number. Also, to accommodate ACSI protocol violations (e.g. the use of partial commands), a controller should recognize a command being sent to another device and immediately relinquish the bus and return to its idle state, i.e. waiting for a new command.

The first byte of a command packet is recognized when A1 is low, R/_W is low and _CS goes low. _CS is generated (automatically) when the DMA Data Port is read or written by the Atari ST. A1 is significant only when a _CS occurs. All peripherals should monitor the bus during this time. When a device recognizes its device number and a valid command, it should enable its bus interface, and assert the interrupt signal to indicate that it has accepted the command and is ready for the next byte (or has a status byte available).

The remaining bytes of the command packet are sent with A1 high, again by the Atari ST asserting _CS for each byte. Readiness to accept the next byte (and therefore receipt of the previous byte) is acknowledged by the peripheral asserting IRQ. The interrupt is cleared by the Atari ST upon writing the next byte. This is the Command phase. After the peripheral has received the last byte, the peripheral proceeds to the data phase (if any) of the command. At the completion of the data phase, a status byte is made available and then interrupt asserted. This is the Status phase. The Atari ST will wait for the interrupt, and then read the status byte by asserting _CS (with A1 high). When the status byte is read, the peripheral should clear the interrupt and get off the bus.

NOTE: With some ACSI peripherals, a 20 microsecond delay is required between the sending of each Command Byte and checking the GPIP register bit 5 for an interrupt. Always allow for a 20 microsecond delay.

During the data input phase (DMA into the Atari ST memory), the peripheral should indicate that a byte is available by asserting _DRQ. The Atari ST acknowledges the acceptance of a byte by asserting _ACK. This clears the previous _DRQ, which in turn logically clears _ACK. In actual implementation, _ACK is always asserted for 250ns.

The Atari ST can DMA up to 127.5K bytes (i.e. 255*512-byte blocks) in a single operation. It is best if DMA input from a peripheral occurs in multiples of sixteen bytes. If that is not the case, multiple commands must be issued (without resetting the DMA controller) to cause the peripheral to generate enough input bytes so that they cross the next 16 byte boundary, thus flushing the FIFO, and are written to Atari ST RAM.

Quirks and Idiosyncrasies

No Message Phase (As compared to SCSI) - SCSI has a "Message Phase" at the completion of a Command Phase (sending all the command bytes). This involves asserting the SCSI _MSG signal and gating it with other signals to allow the Message byte to be returned. ACSI does not permit this. See the example in Chapter 4, DMA Programming Steps, Paragraph 9 "Upon DMA Completion" for an example of how ACSI handles this.

Device Number Selection - When deciding device number selection, remember that the current Atari Hard Disk Interface (AHDI) driver software requires that all hard disk drives use device numbers from 0 up, without any gaps. I.e. hard disk drives are allocated 0, 1, 2 etc. in that order.

Device Number 7 - In order to improve reliability, the SLMC device number should not be set to 7. In general, device #7 should not be used.

DMA CHIP ANOMALY - It is possible to get a double strobe from the DMA chip if the DMA Mode Control Register is not written to immediately after the DMA Data Port. For this reason, while writing to the DMA Data Port it is necessary to write the DMA Mode Control operation for the next operation. This can be done by writing a long word to the Data Port with the data in the upper word and the next operation in the lower word. See the move.l in the programming examples in chapter 4 for an example of how to do this. This restriction only applies to the writing of the DMA Data Port and not to the writing of the DMA Mode Control Register.

Number of DMA Controllers - No more than 4 *controllers* should be connected to the DMA bus as reliability will degrade severely. However, each of these controllers could conceivably serve more than one device. (For example, one hard disk controller may serve multiple drives).

Total Cable Length - Total cable length through the entire DMA Daisy Chain should be <= 72" (182cm).

SLMC Laser Printer Controller - The SLMC Laser printer controller is powered by the SLM Laser printer. If the SLM Laser printer is turned off while using the Atari ST computer the computer may hang or data may be corrupted.

FIFO Flushing-This difference is important to the programmer to guarantee correct and complete data transmission. The Atari ST DMA channel uses high performance FIFOs (First In First Out registers) that need to be flushed at the end of each data transfer to actually put the data in RAM, i.e a FIFO is not necessarily empty at the start of each operation. On output from the host, the presence of these FIFOs generally is not significant, since the DMA channel FIFO will always be filled in multiples of eight words (16 bytes) and the peripheral need not accept all the data. However, on input to the host, the only way to cause the data contained in the FIFOs to be written to RAM is to meet (or exceed) a multiple of 16 bytes. Bytes are only written to RAM in multiples of 16 bytes.

DMA Address - Load the low order byte first, then mid, then high. The DMA pointer must be set before writing to the sector count register.

Daisy Chaining - Controllers which are designed to pass ACSI signals through to another controller in the daisy chain should buffer ACSI signals on the ACSI bus. It is recommended that critical signals such as RESET and ACK be buffered via a 74LS244 or equivalent.

CHAPTER 4: PROGRAMMING THE DMA PORT AND ACSI PERIPHERALS

Software Overview

Commands are sent to the controller via the ACSI command block as defined later in this chapter under **Command Block Overview.** The Atari Computer System Interface (ACSI) command block is much like the Small Computer Systems Interface (SCSI). The command bytes are sent over the DMA channel one byte at a time with the computer and peripheral "talking" to each other between bytes (this is called "handshaking"). Once the last byte is sent and the DMA channel is enabled, the transfer will begin. Once the DMA transfer is complete, the status byte can be read to determine if the operation was successful.

All communication over the DMA channel is done via the registers described below. These registers provide the entire interface to both the DMA controller and the Target device. The application using the DMA channel can select either the internal floppy disk controller (FDC) or the external target device connected to the DMA port.

The interrupt outputs of the internal floppy disk controller and the external ACSI DMA port are logically OR'ed. The pin of the MFP GPIP will read as a '0' if either the FDC or a selected ACSI device controller is asserting its interrupt request.

Atari ST DMA Register Map

```
                           ATARI ST REGISTER MAP

                                 D D                 D
                                 8 7                 0

$FF8600          . . . . . . . . . . . . . . . . .  RESERVED
$FF8602          . . . . . . . . . . . . . . . . .  RESERVED
$FF8604   R/W  . . . . . . . . . x x x x x x x x    DMA Data Register
$FF8606   R    . . . . . . . . . . . . . . x x x    DMA Status Register
                                           | | └─   ERROR
                                           | └───   Block Count Zero
                                           └─────   Data Request Inactive
$FF8606   W    . . . . . . . . x x x x x x x x x    DMA Mode Control
                               | | | | | | | | └─   UNUSED
                               | | | | | | | └───   A1
                               | | | | | | └─────   A2
                               | | | | | └───────   CS (1=external, 0=internal)
                               | | | | └─────────   Block Count Register Select
                               | | | └───────────   0-RESERVED
                               | | └─────────────   0-RESERVED
                               | └───────────────   DRQ (1=internal, 0=external)
                               └─────────────────   DMAOUT

$FF8609   R/W                    x x x x x x x x    DMA Base & Counter High
$FF860B   R/W                    x x x x x x x x    DMA Base & Counter Mid
$FF860D   R/W                    x x x x x x x x    DMA Base & Counter Low
$FFFA01   R                      x x x x x x x x    GPIP General Purpose Input Port
                                     └───────────   IRQ - Interrupt
                                                    Request
```

DMA Mode Control Register Bits ($FF8606)

Bits 9 through 15	Not used.
Bit 8	This bit is used to set the DMA controller to read from or write to the target device. Changing the state of this bit resets the controller. You should reset the DMA controller before each DMA operation. Resetting the controller flushes the internal FIFO and clears the Sector Count Register.
Bit 7	This bit when set (1) gates _DRQ to the internal bus (floppy). When clear (0) _DRQ is gated to the external ACSI bus.
Bits 5 and 6	Reserved.
Bit 4	When this bit is set (1), the DMA controller sector count register can be written to. This is always done immediately following resetting the DMA controller and initializing the DMA Base Counter. The count is in multiples of 512 bytes. The sector count byte sets the upper limit on the number of bytes that can be transfered in a single DMA operation. Up to 127.5Kbytes may be transfered at one time. This maximum byte count is equal to the maximum sector count (FF16 or 25510) multiplied by the block-size of 512.

NOTE: This is the upper limit on the number of bytes to transfer. The lower limit for the Sector Count is 1 for any transfer of 512 bytes or less. For example, if the target device is sending 16 bytes of data then the sector count register should be set to 1.

Bit 3	This bit when set (1) gates _CS to the external bus (ACSI). When clear (0) _CS is gated to the internal bus (floppy).
Bit 2	A2 is not used for DMA operations, except for the internal floppy disk subsystem.
Bit 1	A1 is used to signal the start of a new command block. The start of a new command block happens when this bit is clear (0) and a CS (Chip Select) occurs. For the rest of the command this bit must be set (1). A1 is also used for the floppy disk subsystem.

Bit 0	Not used.
DMA Internal FIFO	Internal to the DMA chip is what logically looks like a sixteen byte FIFO. On DMA reads to the Atari ST this FIFO is only written to the memory of the Atari ST when it is completely filled. Therefore, transfers of less than 16 bytes will remain in the FIFO until it is flushed or filled by subsequent commands. For example, if we are to read 2052 bytes of data from the target device, it will be necessary to set the DMA count register to 5 (5 x 512=2560 bytes). When the transfer is finished, there will still be four bytes left in the FIFO since 2052 is not divisible by 16. To flush the FIFO it is necessary to execute a command that will generate (at least) twelve more bytes. An easy way of doing this is to request two sectors (blocks) from the target device. This way the second block transferred will flush the remaining bytes from the first sector. The data from the second sector may be ignored. You must be sure enough RAM memory is allocated for the extra unused data.
Command/Status Handshake	The command and status bytes are sent or received across the DMA controller's data channel. The Interrupt Request Signal (IRQ) is generated by the target device to indicate that the device is ready to accept another command byte or to indicate that a byte is available. Bit 5 of the General Purpose Input Port (GPIP) of the 68901 Multi-Function Peripheral chip (MFP) is used to indicate that a device s requesting service. This bit will be zero (0) when either the floppy disk controller (FDC) or the target DMA device asserts interrupt. Current TOS software polls this bit rather than using the MFP to actually generate a processor interrupt.

Overview of DMA Programming Steps

The following steps <u>must</u> be followed in order to communicate to a standard DMA device. Leading zeros are included in examples for clarity only; they are not required by the assembler.

Software versus Hardware (DMA) Handshake - The terms "Handshake" and "DMA" are used frequently in this text. Both terms refer to methods of transferring data (DMA or handshake), commands or status (both handshake) over the Atari ST ACSI or Atari ST internal buses. Handshake refers to the process of sending or receiving bytes to or from a peripheral device, one at a time, with intervening "handshaking" requiring host intervention. DMA is completely hardware controlled and does not require processor intervention to transfer data.

Software Handshaking provides a way to transfer bytes, usually command and status bytes, under processor control. The first byte of a command is written with A1=0. Normally, the addressed controller will do some processing of this command, and assert _IRQ to indicate its readiness to take the next byte. The act of writing that byte causes the peripheral to deassert _IRQ, which it will then assert again when it is ready for the next byte. The last byte of a typical command packet that results in a DMA data transfer will not cause _IRQ until the status byte is ready to be read by the processor (also indicating that the DMA phase is complete).

_CS is automatically asserted when the DMA data register ($FF8604) is accessed. The processor should have already latched the desired value of A1. The R/_W signal and the direction of the data bus is automatically controlled by whether the processor reads or writes the DMA data register.

DMA (Direct Memory Access) requires the Atari ST to set-up the DMA Base Counter with the memory address for data transfer, tell the DMA channel how many "blocks" of data are to be transferred and then initiate the DMA. All blocks are then transferred unless an error occurs. The Atari ST recognizes the completion of DMA by monitoring the GPIP register for the peripheral to assert its IRQ. During the transfer a very fast hardware "handshake" takes place with the host asserted _ACK signal and the peripheral controller asserted _DRQ signal. This occurs on each byte without host processor intervention.

NOTE: All Hex numbers below are preceded by a dollar sign ($).

---

1. Set the Floppy Semaphore (flock)

When **flock** is set, the TOS operating system will not access the DMA controller (something that is frequently done at interrupt time).

Example:

```
flock   equu      $43e          ; $43e = flock (Floppy Semaphore) address
        st        flock         ; Lock the dma channel by setting flock
```

2. Change the state of the DMAOUT bit

Change the state of this bit to reset the DMA chip via the DMA Mode Control Register. Usually this involves retaining other previously SET data bits. For example, writing $0190, $0090 to this register toggles the DMAOUT bit leaving it in a Read (0) state, and sets the Sector Count Register Select in preparation for receiving data. Please see the register map in the Software Overview Section for detailed register descriptions.

Example:

```
dma     equ       $ffff8606     ; $ffff8606 = DMA Mode Control Register
```

Read Operation - (Peripheral to Host)

```
        move.w    #$0190, dma   ; SET- Write, Select Sector Count Reg.
        move.w    #$0090, dma   ; SET- Read, Select Sector Count Reg.
```

Write Operation

```
        move.w    #$0090, dma   ; SET- Read, Select Sector Count Reg.
        move.w    #$0190, dma   ; SET- Write, Select Sector Count Reg.
```

3. Load the Sector Count Register

This is a DMA trigger that signals the DMA chip to expect the register value multiplied by 512 bytes of information over the ACSI bus. Each sector (block) to the DMA chip is 512 bytes.

NOTE: The Sector Count Register was selected in the above "Toggle the DMAOUT bit" programming examples. At this time the program is ready to load the Sector Count Register with the number of sectors (blocks) to be read or written.

Example:
```
blkcnt   equ       $xx             ; xx=# of sectors (blocks). Max=255 ($ff)
data     equ       $ffff8604       ; $ffff8604 = Data Reg. will contain Sector Count
         move.w    #blkcnt, data   ; SET- Read, Select Sector Count Reg.
```

NOTE: Refer to the DMA chip anomaly described below. Only when writing to the Sector Count Register is it allowable to use a word write as shown above. On all other Data Port writes use a long write (move. l) with the MSW sent to the DMA Data Port ($ffff8604) and the LSW sent to the DMA Mode Control Register ($ffff8606). See examples below.

4. Write the First Command Byte to the DMA Port

Example:
```
gpip     equ       $fffffa01           ; address of general purpose register
dma      equ       $ffff8606           ; $ffff8606 = DMA Mode Control Register
data     equ       $ffff8604           ; $ffff8604 = Data Reg.
         move. w   #$00000088, dma     ; assert command signal. A1='0'.
         move. l   #$00co008a, data    ; first byte of command, and mode control reg set-
                                       ; up for next byte. c=controller # left shifted one
                                       ; bit (e.g. $C is really controller 6). o=command
                                       ; opcode (not left shifted).

         btst.b    #5, gpip            ; test bit 5 of gpip for IRQ
```

5. Wait for Acknowledge

Once the byte has been sent, wait for an acknowledge from the target device by monitoring bit 5 of the GPIP (General Purpose Interrupt Port of the MC68901). Bit 5 is cleared when the target device is ready for the next byte. This operation should also timeout in case a response is not obtained from the target device. See the programming example above for an example of how to do this.

The following programming example combines a required 20 microsecond delay (between command bytes) with the 3 second timeout delay. This loop can be inserted in the "Write the Command Byte to the DMA Port" programming example above to obtain a complete command byte write sequence. The timeout may not be enough for some peripherals (eg. a tape drive, or CD ROM).

Example:
```
gpip      equ       $fffffa01       ; address of general purpose register
hz200     equ       $4ba            ; system 200Hz timer
          moveq.l   #2, d1          ; approximately 5 millisecond delay
          add.l     hz200, d1       ; timing constant equivalent to 200 hz

del20mic: cmp.l     hz200, d1       ; are we done looping?
          bge       del20mic        ; loop until 20 microsecond delay done

          moveq.l   #600, d1        ; start 3 second timeout on Acknowledge
          add.l     hz200, d1
```

NOTE: The del20mic timing loop is not required for all peripherals.

```
del3sec:  btst.b    #5, gpip        ; test to see if target has acknowledged byte
          beg       exit            ; command byte acknowledged
          cmp.l     hz200, d1       ; loop done?
          bge       del3sec         ; branch not done otherwise error!

          moveq.l   #-1, d1         ; timeout: set error flag in d1 and exit
exit:     rts
```

6. Send the intermediate Command bytes.

Remember to send the byte and the operation for the next byte in one instruction. Wait for acknowledgement from the target device. The programming examples above (including the first command byte example) are also applicable to the next 4 command bytes. If the command you are programming is a six byte command, you will send **four** bytes in this manner.

7. Send the final Command byte. Start DMA if required by the command.

Using a long write to the DMA Data and Mode control port, send the last command byte (usually 0) in the MSW and set (1) or clear (0) bit 7 in the LSW. If bit 7 of the DMA Mode Control Register is set (1) the DMA is on the internal (floppy bus). If bit 7 is cleared (0) the DMA is over the ACSI external bus.

NOTE: This example presumes all commands require DMA

Example:
```
data      equ       $ffff8604         ; $ffff8604 = Data Reg.
gpip      equ       $fffffa01         ; address of general purpose register
```
If DMA is from a target on the ACSI bus use the following instruction:
```
          move.l    #$00??0000,data   ; send last byte and dma over ACSI bus
```
If DMA is to a target on the ACSI bus use the following instruction:
```
          move.l    #$00??0080,data   ; send last byte and dma over ACSI bus

ack:      btst.b    #5, gpip          ; test to see if target has acknowledged byte
          beg       exit              ; command byte acknowledged
```
Place a timeout here. See delay examples above.
```
          bra       ack               ; acknowledge not returned
exit:     rts
```

The ?? above refer to command modifiers that may appear in some commands. See the individual command descriptions for details. These are normally zero.

8. Upon DMA Completion:

Receive the status byte, then clear **flock** so that TOS can reprogram the DMA channel.

Example:
```
data      equ       $ffff8604       ; DMA Data Register
dma       equ       $ffff8606       ; DMA Mode Control and Status Register
gpip      equ       $fffffa01       ; general purpose register
flock     equ       $43e            ; $43e = flock (Floppy Semaphore) address
hz200     equ       $4ba            ; 200 hz delay value

stwait:   btst.b    #5,gpip         ; wait for status byte
          bne       stwait
stbyte:   move.w    #$8a, dma       ; select status register
          move.w    data, d0        ; read status byte
```

NOTE: With some peripherals the following delay is required between commands. This can accommodate the transfer of a message byte between an SCSI controller and an SCSI - ACSI adapter board, for example.

```
          moveq.l   #2, d1          ; approximately 5 millisecond delay
          add.l     hz200, d1       ;
stdel:    cmp.l     hz200, d1
          bge       stdel

exit:     sf        flock           ; Relinquish the dma channel by clearing the flock
```

Command Descriptor Block Overview

The Command Descriptor block shown in the illustration below is an example that is applicable to **some** target devices on the ACSI bus. Device commands by type and op-code are described in Appendix A for the AHDI (Atari Hard Disk Interface), APPI (Atari Page Printer Interface) and the CDAR Audio/ROM CD unit.

Command Descriptor Block

```
       Bit: 7  6  5  4  3  2  1  0
   Byte 00: x  x  x  x  x  x  x  x
            |     |  |__________|_____Operation Code
            |_____|___________________Controller Number

   Byte 01: .  .  .  .  .  .  .  .    Most Significant


   Byte 02: .  .  .  .  .  .  .  .

   Byte 03: .  .  .  .  .  .  .  .    Least Significant byte of the Logical
                                      Address

   Byte 04: x  x  x  x  x  x  x  x    Operation Length (Usually Sector count in
                                      512 Byte increments)

   Byte 05: x  x  .  .  .  .  .  .
            |__|______________________Operation Modifiers
```

_DRQ
(Data Request): Used by the Peripheral (Target) to request a DMA data transfer. Direction of transfer is determined by the **R/_W** signal.

Controller Number
A 3-bit number designating the physical controller receiving the Command Block. Up to eight (0 through 7) controllers can be connected to the Atari DMA channel. Usually set on target device with dip switches or jumpers.

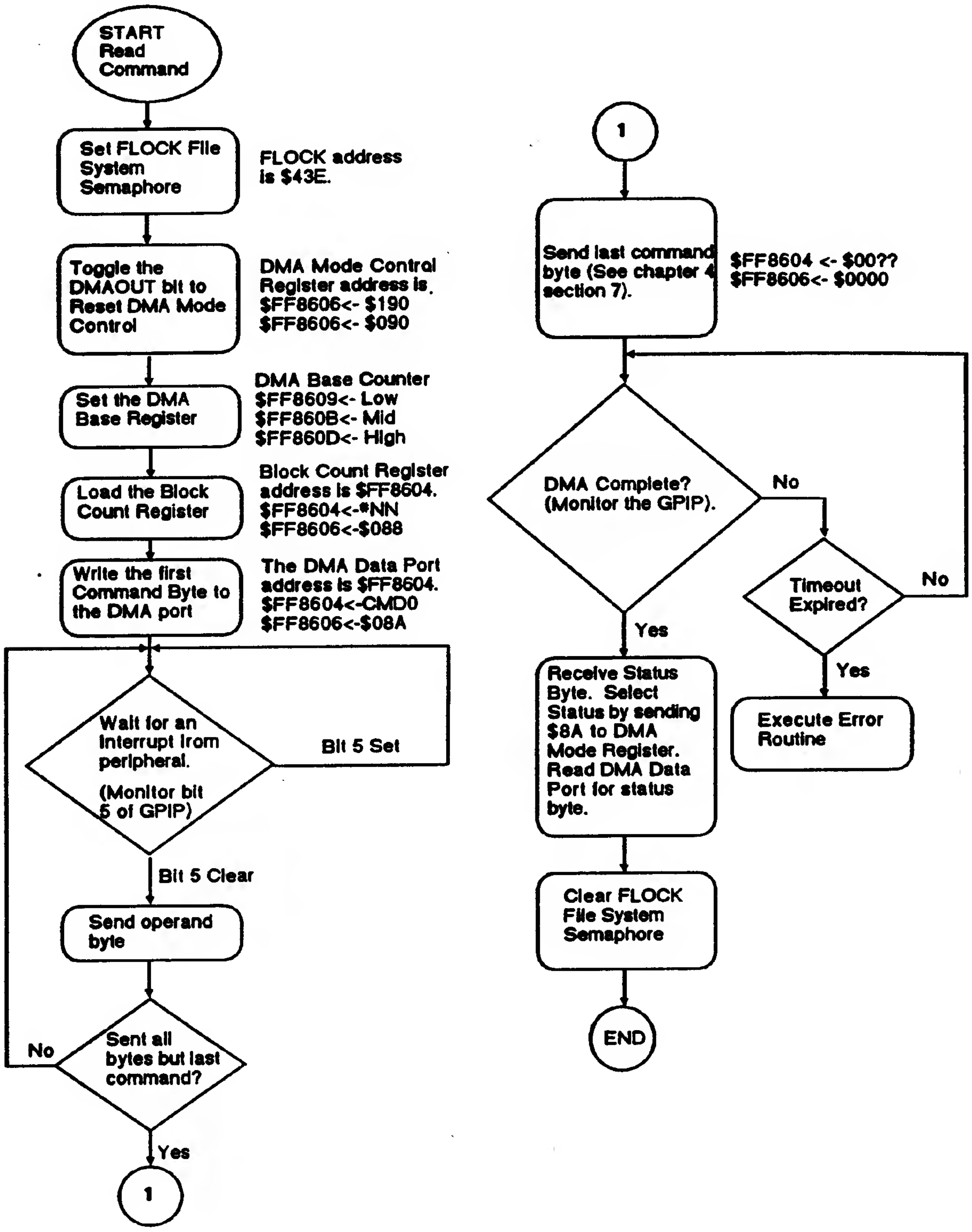
Operation Code
A 5-bit number specifying the operation (Op-code) to be performed by the target controller given the information within the Command Block.

Operation Length (Block Count)
An 8-bit number that determines the number of Blocks (sometimes called "Sectors" or "Data Sectors") to be transferred for read and write commands (should be non-zero). This number can be used to signify lengths in bytes for some command types.

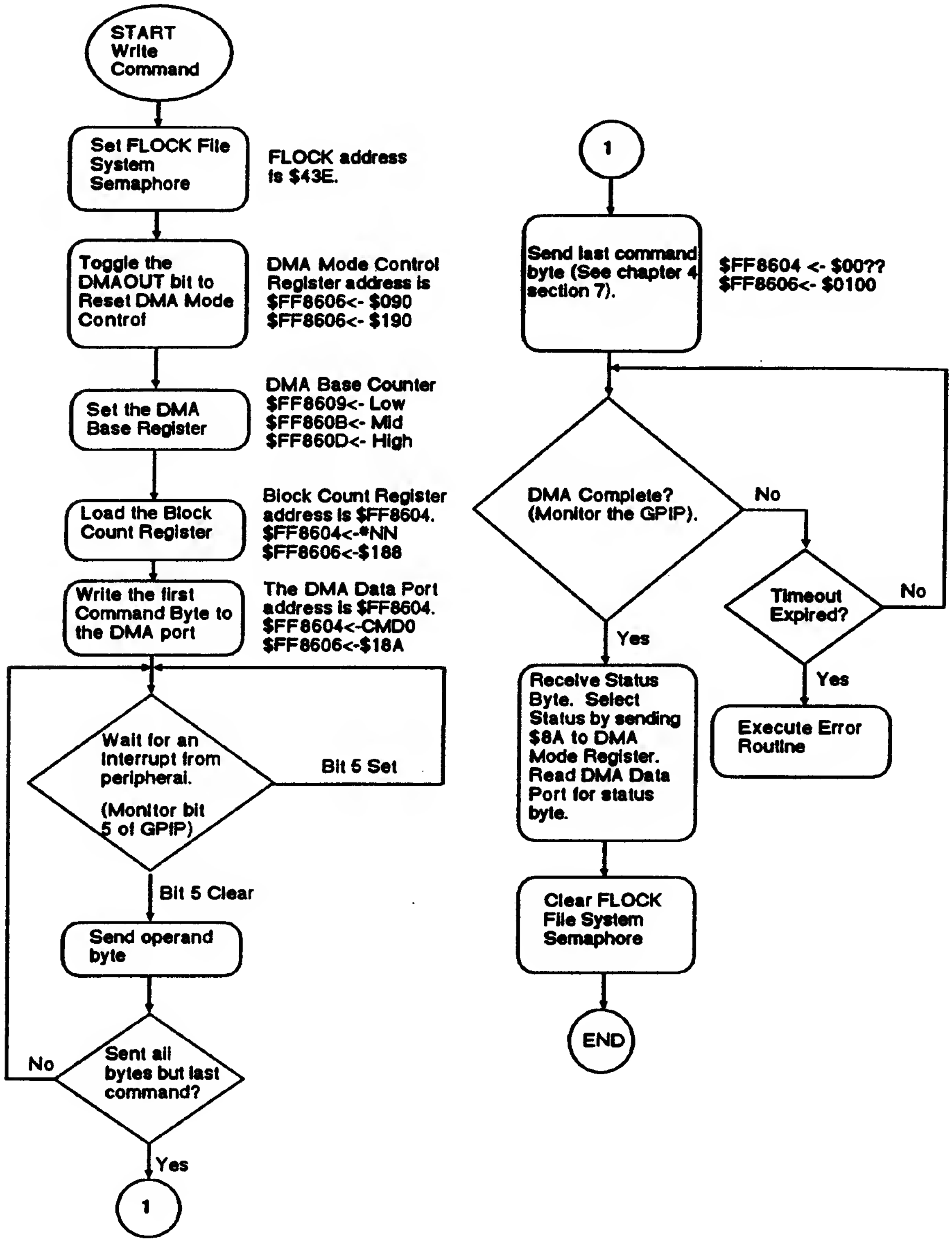
Operation Modifiers

Some commands have optional "Operation Modifiers" that change the way a command is interpreted by the target device. See detailed command descriptions for more information.

Detailed Command-Data-Status Flowchart (Read)


START Read Command
Set FLOCK File System Semaphore
FLOCK address is $43E.
Toggle the DMAOUT bit to Reset DMA Mode Control
DMA Mode Control Register address is.
$FF8606<- $190
$FF8606<- $090
Set the DMA Base Register
DMA Base Counter
$FF8609<- Low
$FF860B<- Mid
$FF860D<- High
Load the Block Count Register
Block Count Register address is $FF8604.
$FF8604<-#NN
$FF8606<-$088
Write the first Command Byte to the DMA port
The DMA Data Port address is $FF8604.
$FF8604<-CMD0
$FF8606<-$08A
Wait for an interrupt from peripheral. (Monitor bit 5 of GPIP)
Bit 5 Set
Bit 5 Clear
Send operand byte
Sent all bytes but last command?
No
Yes
1
1
Send last command byte (See chapter 4 section 7).
$FF8604 <- $00??
$FF8606<- $0000
DMA Complete? (Monitor the GPIP).
No
Yes
Timeout Expired?
No
Yes
Receive Status Byte. Select Status by sending $8A to DMA Mode Register. Read DMA Data Port for status byte.
Execute Error Routine
Clear FLOCK File System Semaphore
END

Detailed Command-Data-Status Flowchart (Write)


START Write Command
Set FLOCK File System Semaphore
FLOCK address is $43E.
Toggle the DMAOUT bit to Reset DMA Mode Control
DMA Mode Control Register address is $FF8606<- $090 $FF8606<- $190
Set the DMA Base Register
DMA Base Counter $FF8609<- Low $FF860B<- Mid $FF860D<- High
Load the Block Count Register
Block Count Register address is $FF8604. $FF8604<-#NN $FF8606<-$188
Write the first Command Byte to the DMA port
The DMA Data Port address is $FF8604. $FF8604<-CMD0 $FF8606<-$18A
Wait for an interrupt from peripheral. (Monitor bit 5 of GPIP)
Bit 5 Set
Bit 5 Clear
Send operand byte
Sent all bytes but last command?
No
Yes
1
1
Send last command byte (See chapter 4 section 7).
$FF8604 <- $00?? $FF8606<- $0100
DMA Complete? (Monitor the GPIP).
No
Yes
Timeout Expired?
No
Yes
Execute Error Routine
Receive Status Byte. Select Status by sending $8A to DMA Mode Register. Read DMA Data Port for status byte.
Clear FLOCK File System Semaphore
END

CHAPTER 5: THE INQUIRY COMMAND

The Inquiry Command - its purpose

The Inquiry command (Opcode 0x12) is used to determine whether a device is attached to the Atari ST and, if so, to return detailed information about it.

Each device that is designed to respond to the Inquiry command returns information specific to that device only. Device specific Inquiry command information is treated separately in Appendix A. Because the Inquiry command is used to determine system configuration, it is essential that the command have the same, six byte, format for **all** devices. This section describes how to use Inquiry to determine whether a device is attached to the Atari ST.

A program should <u>never</u> assume that a particular device (e.g. a laser printer) has a fixed device number. <u>All</u> programs should consistently use the Inquiry command to determine the presence of the required device, and its configured device number.

Using Inquiry

The Inquiry command has several idiosyncrasies that need to be considered when using the command to obtain device configuration or to determine device availability. They are:

1. Some implementations of the AHDI (Atari Hard Disk Interface) do not recognize the Inquiry command and, if present, return an error. The fact that an error (Invalid Command) **is** returned may indicate that the AHDI is present. If no _IRQ is received after a delay of approximately 3 seconds it may be assumed that the AHDI is not configured at the specified device number.

2. Some devices require that the 16 bytes of response to the Inquiry command be returned using DMA and others require a "handshake" for each byte. In general, a DMA response is preferred, and at the time of publication only the Atari Page Printer Interface requires the handshake approach. It sends the status byte first, and then expects you to handshake the response bytes.

3. The CDAR CD ROM supports DMA, and will therefore try to DMA the response.

4. The APPI (Atari Page Printer Interface) returns an invalid command status if an attempt is made to DMA the 16 status bytes.

The steps that **must** be followed when using the Inquiry Command are below.

1. Starting with Device 0 try to perform the Inquiry command using DMA to transfer the 16 status bytes. If an error status is returned or if no status is returned (because the 3 second delay times out) attempt to handshake the status bytes after reissuing the Inquiry command. If an error occurs when trying to handshake the first byte, the device attached to that device address is the AHDI (Atari Hard Disk Interface). Only the AHDI (hard disk) may give an error on both DMA and handshake. All other devices should respond with 16 bytes of status to either the DMA or handshake requests.

2. Continue as in step one for the remaining seven device addresses. Byte zero of all commands contains the device (controller) number. If a device is there, all six command bytes will be accepted. If no device is connected at the address to which a command is sent, the 3 second timeout used between sending a command byte and receiving its acknowledge, will expire. This timeout will occur on the very first command byte. The occurrence of a timeout indicates that a device was not connected at the device address to which the command was sent.

Inquiry Command Usage Chart

The chart below indicates the actions of each of several supported device types.

<u>Laser Printer</u>	
DMA Inquiry Response	ERROR (Invalid Command)
Handshake Inquiry Response	If Byte 5 bit 7 is set, the ID is sent back. If not set, an error has occurred. A null status byte is returned before the Inquiry data comes back.
<u>Hard Disk</u>	
DMA Inquiry Response	ERROR (Invalid Command) *
Handshake Inquiry Response	ERROR (Invalid Command) **
<u>CD-ROM</u>	
DMA Inquiry Response	DMAs 16 bytes of CD-ROM identification data. (Not all bytes contain information.)
Handshake Inquiry Response	Don't try to Handshake with the CD-ROM. It will attempt to DMA anyway. **

* Some hard disk controllers will correctly return data from Inquiry. (They may not, however, return the 16 or more bytes to cause the FIFO to be written to RAM in one operation.)

** For <u>any</u> device, try the standard DMA form first. If it succeeds, do <u>not</u> try the handshake form.

If the device does not DMA or handshake DMA data, the existence of the device can be confirmed by reading sector 0.

APPENDIX A
DEVICE COMMANDS AND DRIVERS

A1: ACSI-AHDI Commands

The AHDI command set contains eight general purpose hard disk operations. Please note that there is no zero checking (error returned if other than zeros found) of unused command descriptor block fields since there are command fields reserved for future enhancement. OpCodes occupy the least significant 5 bits of the byte. The following table is a summary of available command opcodes.

OpCode	Command
$00	Test Unit Ready
$03	Request Sense
$04	Format Drive
$08	Read*
$0A	Write*
$0B	Seek
$12	Inquiry

* Multisector transfer with implied seek.

NOTE: All ACSI devices *must* support the Test Unit Ready ($00), Read ($08), and Inquiry ($12) commands.

$00 Test Drive Ready

This command returns zero status if the requested drive is powered on and ready. If the drive is not ready, the Check Condition bit is set in the Completion Status Byte.

```
Byte 00: x  x  x  0  0  0  0  0
         |     |  |__________|_____Test Drive Ready OpCode
         |_____|___________________Controller Number

Byte 01: 0  0  0  0  0  0  0  0

Byte 02: 0  0  0  0  0  0  0  0

Byte 03: 0  0  0  0  0  0  0  0

Byte 04: 0  0  0  0  0  0  0  0

Byte 05: 0  0  0  0  0  0  0  0
```

$03 Request Sense

This command returns a four-byte block containing the status error code associated with the immediately preceding drive operation (see **Status Structure** and **Status Error Codes**).

```
Byte 00: x   x   x   0   0   0   1   1
         |       |   |_______________|_____Request Sense OpCode
         |_______|_________________________Controller Number

Byte 01: 0   0   0   0   0   0   0   0

Byte 02: 0   0   0   0   0   0   0   0

Byte 03: 0   0   0   0   0   0   0   0

Byte 04: 0   0   0   0   0   1   0   0

Byte 05: 0   0   0   0   0   0   0   0
```

Status Structure (Completion Status Byte)

The Completion Status Byte is returned following the successful (or unsuccessful) execution of a command. If an error occurs, the Check Condition bit is set, so if Check Condition = 1, there is an error, but if Check Condition = 0, no error exists. Further information can be obtained by a subsequent Request Sense command. (The Check Condition bit is never set in response to a Request Sense command).

```
Byte 00: 0   0   0   0   0   0   x   0
                                 |_______Check Condition
```

Status Error Codes

The following status block is returned by the Request Sense command; it must be at least four bytes long, the first byte of which is the most important. Specific peripherals can support larger status blocks; the Atari CD-ROM, for example, returns 16 bytes. See **Error Codes and Descriptions** for full explanations of the error codes.

```
Byte 00: 0 x x x x x x x
           |___________|____Error Code (Most important byte)

Byte 01: 0 0 0 0 0 0 0 0

Byte 02: 0 0 0 0 0 0 0 0

Byte 03: 0 0 0 0 0 0 0 0
```

NOTE: For some devices, (e.g. Atari hard disks) this DMA is only four bytes long, so to get data into ST RAM requires performing another DMA input operation to generate enough bytes to cross the FIFO 16-byte boundary without toggling DMAOUT.

$04 Format Drive

The controller writes from index to index all ID and data fields with a block size as specified by an immediately previous Mode Select command. If no Mode Select command is executed, the previous data block size is used. Data fields are completely written with $6C unless the Data Pattern Flag bits are set. The Interleave Factor is equivalent to the number of disk revolutions required to sequentially read one track in one operation.

```
Byte 00: x  x  x  0  0  1  0  0
         |     |  |__________|_____Format Drive OpCode
         |_____|_____________________Controller Number

Byte 01: 0  0  0  0  0  0  0  0


Byte 02: 0  0  0  0  0  0  0  0


Byte03:  0  0  0  0  0  0  0  0

Byte 04: x  x  x  x  x  x  x  x
         |____________________|_____Interleave Factor

Byte 05: 0  0  0  0  0  0  0  0     Unused
```

$08 Read Command

This command transfers to the host the specified number of blocks starting at the specified logical starting block address.

```
Byte 00: x  x  x  0  1  0  0  0
         |     |  |__________|_____Read OpCode
         |_____|_____________________Controller Number

Byte 01: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address High

Byte 02: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Mid

Byte 03: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Low

Byte 04: x  x  x  x  x  x  x  x
         |____________________|_____Block Count

Byte 05: 0  0  0  0  0  0  0  0
```

$0a Write Command

This command transfers to the drive the specified number of blocks starting at the specified logical starting block address..

```
Byte 00: x  x  x  0  1  0  1  0
         |     |  |__________|_____Write OpCode
         |_____|___________________Controller Number

Byte 01: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address High

Byte 02: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Mid

Byte 03: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Low

Byte 04: x  x  x  x  x  x  x  x
         |____________________|_____Block Count

Byte 05: 0  0  0  0  0  0  0  0
```

$0b Seek Command

This command causes the selected drive to seek to the specified logical starting block address.

```
Byte 00: x  x  x  0  1  0  1  1
         |     |  |__________|_____Seek OpCode
         |_____|___________________Controller Number

Byte 01: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address High

Byte 02: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Mid

Byte 03: x  x  x  x  x  x  x  x
         |____________________|_____Logical Block Address Low

Byte 04: 0  0  0  0  0  0  0  0

Byte 05: 0  0  0  0  0  0  0  0
```

A2: Error Codes and Descriptions

Drive Errors	Description
0x00	No Error (aggregate command complete)
0x01	No Index (Index pulses were not detected)
0x03	Write Fault
0x04	Drive Not Ready (allow 20 seconds for drive spinup)
0x06	No Track 00

Controller Errors	Description
0x10	ID ECC Error
0x11	Uncorrectable Data Error (ECC shifts exceeded sector length)
0x12	ID Address Mark Not Found
0x13	Data Address Mark Not Found
0x14	Record Not Found (sector number nonexistent)
0x15	Seek Error (recalibrated to track 00)
0x18	Data Check in No Abort Mode (ECC check abort disabled)
0x19	ECC Error During Verify (request ECC correction pattern)
0x1A	Bad Block (formatted with bad sector mark)
0x1C	Unformatted or Bad Format (miscellaneous format errors)

Command Errors	Description
0x20	Invalid Command (opcode not implemented)
0x21	Invalid Address
0x23	Volume Overflow
0x24	Invalid Argument
0x25	Invalid Drive Number
0x26	Byte Zero Parity Check

Miscellaneous Error	Description
0x30	Controller Self Test Failed (cannot be ignored)

Specific Applications

For more information about existing ACSI applications, see the following documents:

"The Atari Page Printer Interface", Atari Corporation

"CDAR504 Developer Reference Manual", Atari Corporation

"TT030 TOS Release Notes", Atari Corporation

"Atari TT030 Hardware Reference Manual", Atari Corporation

APPENDIX B
CABLES AND SCHEMATICS

B1: ACSI Bus Diagram

The following diagram shows the ACSI to AHDI port implementation:

```
Host (DB19S)                              Target
        1 <-----------D0------------------>
        2 <-----------D1------------------>
        3 <-----------D2------------------>
        4 <-----------D3------------------>
        5 <-----------D4------------------>
        6 <-----------D5------------------>
        7 <-----------D6------------------>
        8 <-----------D7------------------>
        9  -----------_CS----------------->
       10 <-----------_IRQ----------------
       11  -----------GND-----------------
       12  -----------_RST---------------->
       13  -----------GND-----------------
       14  -----------_ACK---------------->
       15  -----------GND-----------------
       16  -----------A1------------------>
       17  -----------GND-----------------
       18  -----------R/_W---------------->
       19 <-----------_DRQ----------------
```

B2: ACSI Signal Specification

Mnemonic	Name	Characteristics
_RST	Reset	TTL levels, active low
A1	Address 1	TTL levels, active high
_IRQ	Interrupt Request	TTL levels, active low, 1K ohm pullup on host computer side
_CS	Chip Select	TTL levels, active low
R/_W	Read/Write	TTL levels, high = read
_DRQ	Data Request	TTL levels, active low, 1K ohm pullup on host computer side
_ACK	Acknowledge	TTL levels, active low
DATA	Data Bus (0-7)	TTL levels, active high

NOTE: It is recommended that padding resistors are used for ACSI signals (see B3: Example ACSI Adapter Schematics).

ACSI to SCSI Adapter

Description
The ACSI to SCSI adapter provides an interface from the ACSI 19 pin connector to a 50 pin SCSI connected to a single SCSI peripheral (SCSI ID set to '0').

The SCSI cable consists of open collector active low signals. These signals are driven by 7438s which have the capability to drive a doubly terminated SCSI cable. All other components are LS TTL (except the PAL).

Operation
The host, connected to ACSI-IN, will issue a command by driving the data bus with the command in the lower five bits and the device number in the upper three bits. The adapter will be selected if the upper three bits of the command match the setting of the three position dip switch (SW1).

On selection both SD0- and SSEL- will be active until either SBSY- becomes active or the host issues a new command (due most likely to a time-out). After the SCSI device responds with SBSY- the SCSI data bus will be driven with the command byte. The upper three bits will be masked off until the second command byte. The interrupt request (IRQ-) will not be active until the second SREQ-. The IRQ- signal will remain active until a chip select has been issued from the host.

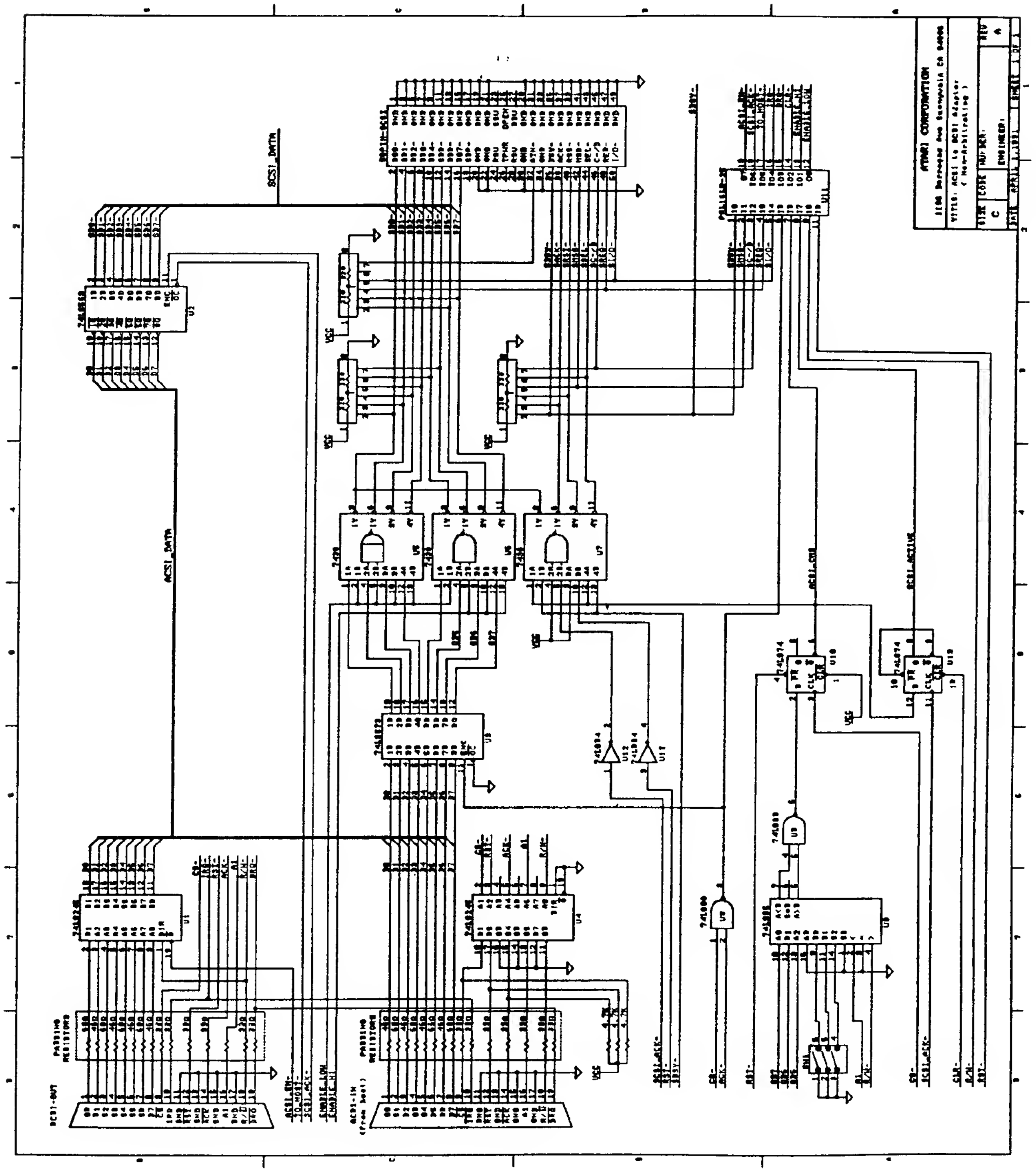
In the data phase IRQ- is held inactive and DRQ- will be active until an ACK-. The SCSI peripheral will control the number of bytes that are transferred and the data rate.

Following the data phase is the status phase. This is byte used to indicate a check condition. If no errors occurred this byte is '00'.

After the status phase is the message phase. This byte is not used by ACSI peripherals and is not sent to the host. An SACK- is issued without an IRQ-.

Following the message byte the SCSI peripheral will release the SBSY- signal and be ready for the next command.

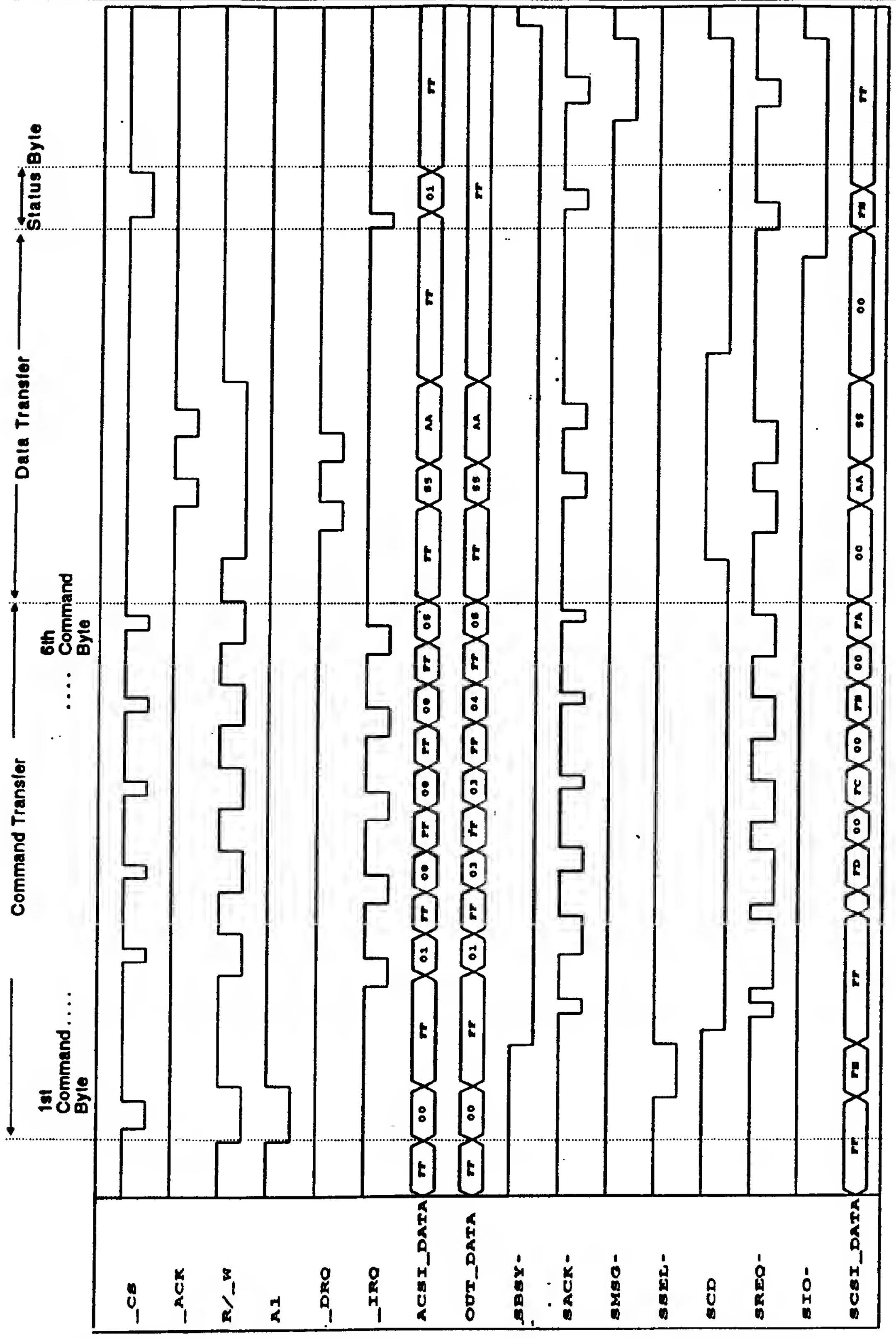
B3: Example ACSI Adapter Schematics



B3: Example ACSI Adapter PAL Equations(cont.)

```
SBSY- SMSG- SCD SREQ- SIO- CS_ACK ASCI_CMD SCSI_ACTIVE RW- GND
RST- /ENABLE_LOW /ENABLE_HI /CLR- /DRQ- /IRQ- /TO_HOST- /SCSI_ACK-
/ACSI_EN- VCC
    ;
    ;      ------------------------------------------------------------
    ;      | SCD  | SIO-   | SMSG- | Direction of Transfer              |
    ;      ------------------------------------------------------------
    ;      |  0   |  1     |   1   | Command Phase (Host -> Target)     |
    ;      ------------------------------------------------------------
    ;      |  1   |  1     |   1   | Data Out Phase (Host -> Target)    |
    ;      ------------------------------------------------------------
    ;      |  1   |  0     |   1   | Data In Phase (Target -> Host)     |
    ;      ------------------------------------------------------------
    ;      |  0   |  0     |   1   | Status Phase  (Target -> Host)     |
    ;      ------------------------------------------------------------
    ;      |  0   |  0     |   0   | Message Out Phase (Target -> Host) |
    ;      ------------------------------------------------------------
    ;

   IF(VCC) ACSI_EN- =
                  /RW- + /SCSI_ACTIVE ; ENABLE '245 IF
                                      ; RW- IS '0' OR IF
                                      ; SCSI DEVICE HAS NOT
                                      ; BEEN SELECTED

   IF(VCC) SCSI_ACK- =
                  /SCD*SIO-*/SREQ-*/SCSI_ACTIVE ; COMMAND OPCODE
               +  /SCD*/SREQ-* SCSI_ACTIVE*CS_ACK*/IRQ-
                                                ; REMAINING COMMAND BYTES
               +  SCD*/SREQ-*CS_ACK*SCSI_ACTIVE*/DRQ-
                                                ; DATA BYTES
               +  /SCD*/SIO-*/SREQ-*/SMSG-      ; MESSAGE BYTE
               + SCSI_ACK-*/SREQ-               ; SACK- HELD UNTIL SREQ-
                                                ; IS INACTIVE
               + SCSI_ACK-*CS_ACK               ; OR CS_ACK INACTIVE

  IF(VCC) TO_HOST- = SCSI_ACTIVE*RW-*SCD        ; DATA BYTES
               +   SCSI_ACTIVE*RW-*CS_ACK       ; STATUS BYTE

  IF(SCSI_ACTIVE*/SCD) IRQ- =
                  /SREQ-*SMSG-*/CSACK           ; COMMAND IRQ's
                                                ; AND STATUS IRQ's

  IF(SCSI_ACTIVE*SCD) DRQ- =
                  /SREQ-*/SCSI_ACK-*/CSACK     ; DRQ's FOR DATA

  IF(VCC) CLR- = /SBSY-*/SCD*/SIO-*SMSG-*/CS_ACK*SCSI_ACK-
                                               ; CLR IS ACTIVE FROM
                                               ; STATUS UNTIL RISING
                                               ; EDGE OF SBSY-
               +  /SBSY-*CLR-
               +  /RST-

  IF(VCC) ENABLE_HI = /SIO-     +   /SCSI_ACTIVE ; ACTIVE HIGH

  IF(VCC) ENABLE_LOW = /SIO-    +   SBSY-       ; ACTIVE HIGH
```

B3: Example ACSI Adapter Example Timing (cont.)



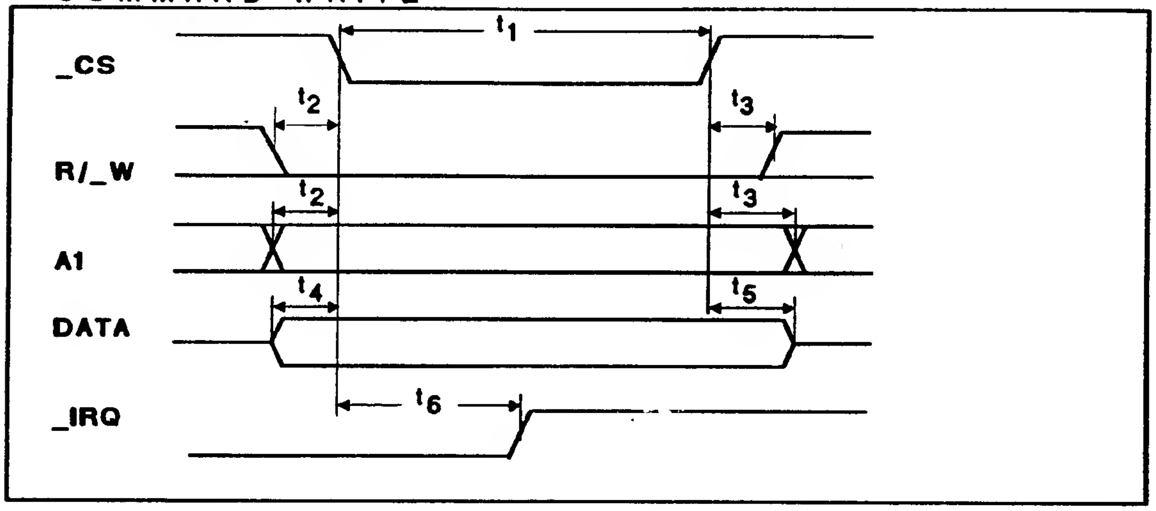
B4: Timing Diagrams

Timing Parameters

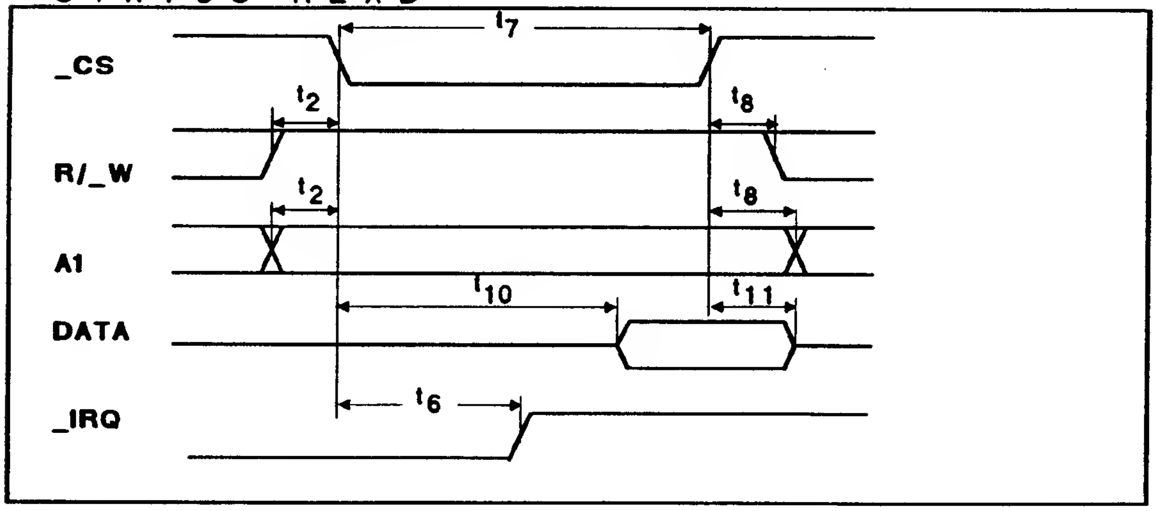
Symbol	Parameter	Min.	Typ.	Max.	Units	Notes
t_1	_CS active for write	220	250	270	ns	
t_2	A1, R/_W setup to _CS	100	125		ns	
t_3	A1, R/_W hold from _CS for write	100	125		ns	
t_4	DATA setup to _CS	80	100		ns	
t_5	DATA hold rom _CS	10			ns	
t_6	_CS low to _IRQ inactive		30		ns	The Laser printer is an exception
t_7	_CS active for Read	220	325		ns	
t_8	Hold time for A1, R/_W from _CS	10			ns	
t_{10}	_CS valid to DATA valid (read)			100	ns	
t_{11}	DATA hold from _CS high	10			ns	
t_{12}	_ACK low time (Read or Write)	220	250	270	ns	
t_{13}	DATA-out setup to _ACK low	20			ns	
t_{14}	DATA-out hold from _ACK high	10			ns	
t_{15}	_ACK low to _DRQ high			180	ns	
t_{16}	_ACK low to DATA-in valid			100	ns	
t_{17}	DATA-in hold from _ACK high	20			ns	

B4: Timing Diagrams - Command/Status

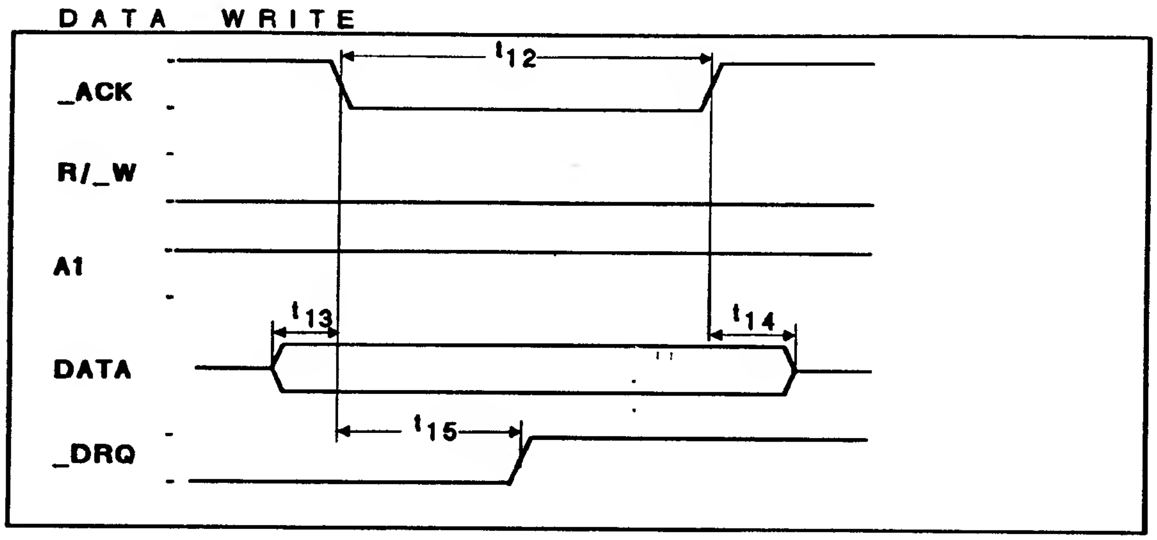
COMMAND WRITE

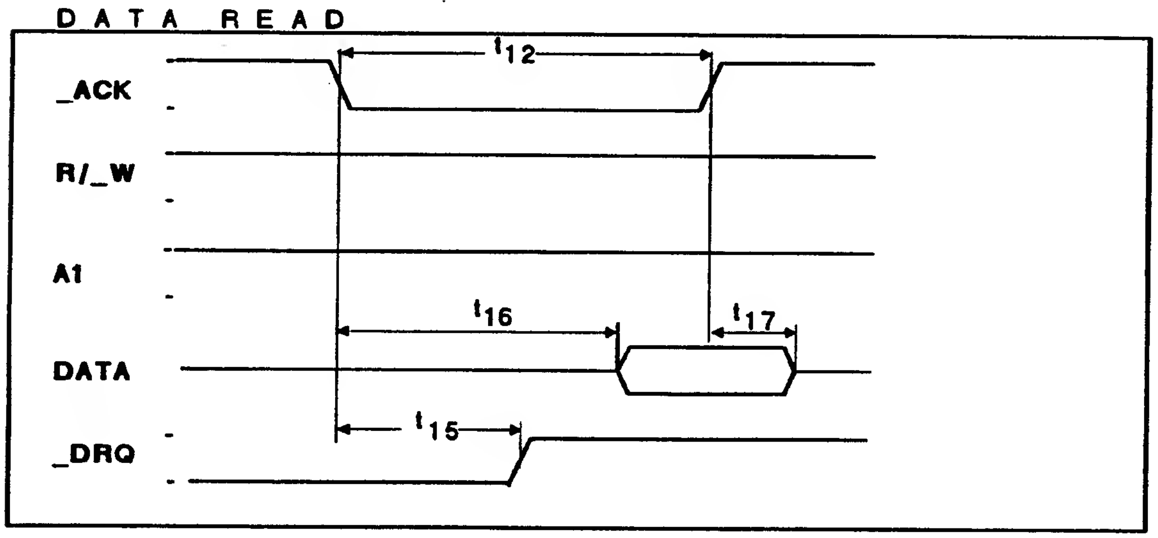


STATUS READ



B4: Timing Diagrams - Data Transfer


DATA WRITE
_ACK
R/_W
A1
DATA
_DRQ
t12
t13
t14
t15


DATA READ
_ACK
R/_W
A1
DATA
_DRQ
t12
t16
t17
t15